A Latent Source Model for Online Time Series Classfication

Zhang Zhang
zsquared@math.umd.edu
University of Maryland, College Park

Advisor
Prof. Aravind Srinivasan
srin@cs.umd.edu
Department of Computer Science
University of Maryland, College Park

May 22, 2014

Abstract

We study a binary classification problem with infinite time series having more than two labels (”event” and ”nonevent” or ”trending” and ”non-trending”). We want to predict the label of the time series given some training data set. Intuitively, the longer we wait, the longer the time series we can observe so that the prediction is more accurate. However, in many applications, such as predicting which topic will go popular in a social network or revealing an imminent market crash, making a prediction as early as possible is highly valuable.Motivated by these applications, we look into a latent source model which is a nonparametric model to predict the binary status of a time series. Our main assumption is that these time series only have a few ways to reach the binary status such as Twitter topic going trending online. The latent source model naturally leads to a weighted majority voting as the classification rule without knowing the latent source structure. In the project: 1. We will investigate the theoretical performance guarantees of the latent source model; 2. We will implement the model by programming language C; 3. We will investigate the strategy to estimate the values of different parameters; 4. We will test our implementation and use the model to predict which news topics on Twitter will go viral to become trends and analyze the results.

1 Introduction

1.1 Motivation

Detection, classification, and prediction of events in the streams of information are important and interesting problems in science and engineering. From detecting a ”fail” in the operation system, to predicting a huge stock index spike, to revealing whether topics in a social network will go popular, extracting useful information from time to time dependent data is fundamental for making decisions.

Different from the 20th century, in the big data era, we have the access to large amount of data related to every human endeavor. Therefore, we are eager to find good ways to analyze the data and make decisions given the data. In social network, there are massive streams of user generated date which will reveal interesting facts, such as blogs and tweets, as well as data from portable electronic devices. These data provides us an amazing opportunity to learn the dynamics of human behavior in the communication online. "How do people make decisions? Who are they influenced by? How do ideas and behaviors spread and evolve?" These are questions that have been impossible to study empirically at scale until recent times.

Big data presents both opportunities and challenges. Big data can reveal the hidden underlying structure in a process of interest. On the other hand, given current computing and storage technology, making computations over so much data at scale is challenge. Fortunately, advances in distributed computing have made it easier than ever to exploit the structure in large amounts of data to do inference at scale. In this project, we do use Mapreduce method to preprocess the big data generated by Twitter. It is about 500G per day.

All of these examples we mentioned above share some common features. There exist some underlying process with specific properties or feature generating these time series such as stock index time series. Looking into time series, we may infer many information, such as detecting anomalous events, predicting the trends of the time series at some future point.

This is difficult to do in general. Many real-world processes defy description by simple models. Here is an paragraph I found in Dr. Nikolov's paper

> *Eugene Wigner's article 'The Unreasonable Effectiveness of Mathematics in the Natural Sciences' examines why so much of physics can be neatly explained with simple mathematical formulas such as $f = ma$ or $e = mc^2$. Meanwhile, sciences that involve human beings rather than elementary particles have proven more resistant to elegant mathematics. Economists suffer from physics envy over their inability to neatly model human behavior. An informal, incomplete grammar of the English language runs over 1700 pages. Perhaps when it comes to natural language processing and related fields, we are doomed to complex theories that will never have the elegance of physics equations. But if that's so, we should stop acting as if our goal is to author extremely elegant theories, and instead embrace complexity and make use of the best ally we have: the unreasonable effectiveness of data.*

In this project, we study the problem of prediction in a complex system using large amounts of data. Specifically, we focus on applying a latent source model to do a binary classification of time series given the big enough training data set (sufficient historical examples). We apply this to the problem of trending topic detection on Twitter and show that we can reliably detect trending topics before they are detected as such by Twitter. At the same time, we aim to introduce a more general setting for doing inference in time series based on a large amount of historical data.

2 Classification Method

In this project, we are applying a setting for model specification and selection in supervised learning based on *a latent source model*. In this setting, the model is specified by a small collection of unknown *latent sources*. We assume that the data were generated by the latent sources.

2.1 Weighted Majority Voting

Weighted Majority Voting model is an important model in machine learning and we apply this model on time series in this project. Suppose we have a time series s: $\mathbb{T} \to \mathbb{R}$ that we want to classify as having either label +1 or -1. Here, each time series is represented as a function mapping $\mathbb{T}$ to $\mathbb{R}$, where we index time using $\mathbb{T}$ for convenience. We denote the sets of all time series with labels +1 and -1 as $\mathcal{R}_+$ and $\mathcal{R}_-$.

In this project, each $r \in \mathcal{R}_+$ gives a weighted vote $e^{-\gamma d^{(T)}(r,s)}$ on whether time series s has label +1, where $d^{(T)}(r,s)$ is some measure of similarity between the two time series r and s. $d^{(T)}(r,s)$ is chosen as the Euclidean distance between s and r in this project. (Similarity measure: the larger the more similar the signals; whereas for a distance: the smaller the more similar). (T) denotes the first T time steps of s, and the constant $\gamma > 0$ is a scaling parameter that determines the influence of each r. Similarly, each negatively-labeled example in $\mathcal{R}_-$ also casts a weighted vote on whether time series s has label -1. Essentially, each example time series in the training data says "'The time series looks like me"' with certain confidence.

In this project, we use squared Euclidean distance between time series as the similarity measure: $d^{(T)}(r,s) = \sum_{t=1}^{t}(r(t) - s(t))^2 = \|r - s\|_T^2$. Notice that this similarity measure only uses first T time steps of example time series r. Since time series in our training data are known, we would not restrict only the first T time steps but instead use the following similarity measure:

$$d^{(T)}(r,s) = \min_{\Delta \in \{-\Delta_{max},\dots,0,\dots,\Delta_{max}\}} \sum_{t=1}^{T}(r(t+\Delta) - s(t))^2 = \min_{\Delta \in \{-\Delta_{max},\dots,0,\dots,\Delta_{max}\}} \|r * \Delta - s\|_T^2 \quad (1)$$

where we minimize over integer time shifts and the maximum time shift $\Delta_{max} \geq 0$.

Finally, we sum up all of the weighted +1 votes and then all of the weighted -1 votes. The label with the majority of overall weighted votes is declared as the label for s:

$$\hat{L}^{(T)}(s,\gamma) = \begin{cases} + & \text{if} \quad \sum_{r\in\mathcal{R}_+} e^{-\gamma d^{(T)}(r,s)} \geq \sum_{r\in\mathcal{R}_-} e^{-\gamma d^{(T)}(r,s)} \\ - & \text{otherwise} \end{cases} \quad (2)$$

Using a larger time window size T corresponds to waiting longer before we make a prediction. We need to trade off how long we wait and how accurate we want our prediction.

2.2 A Latent Source Model

Dr. Chen proposed the latent fource model for nonparametric time series classification. We assume there are m unknown latent sources(time series) that generate observed time series. We denote the set of all such latent sources by $\mathcal{V}$ and each latent source in $\mathcal{V}$ has a true label +1 or -1, which corresponding trending topic or non trending topic. Let $\mathcal{V}_+ \in \mathcal{V}$ be the set of latent sources with label +1, and $\mathcal{V}_-$ be the set of those with label -1. The observed time series are generated from latent sources as follows:
1. Sample a latent source V from $\mathcal{V}$ uniformly at random. Let $L \in \{+1, -1\}$ be the label of V.
2. Sample integer time shift Δ uniformly from $\{0, 1, ..., \Delta_{max}\}$.
3. Output time series S to be the latent source V advanced by Δ time steps, followed by adding noise signal E, i.e., $S(t) = V(t + \Delta) + E(t)$ for $t \geq 1$. Entries of noise E are i.i.d. zero-mean sub-Gaussian with parameter σ, which means that for any time index t,

$$\mathbb{E}[exp(\lambda E(t))] \leq exp(\frac{1}{2}\lambda^2\sigma^2) \quad \text{for all} \quad \lambda \in \mathbb{R} \tag{3}$$

If we know the latent sources and if noise entries $E(t)$ were i.i.d. $\mathcal{N}(0, \frac{1}{2\gamma})$ across t, then the maximum a posteriori estimate for label L given an observed time series $S = s$ is

$$\hat{L}^{(T)}_{MAP}(s;\gamma) = \begin{cases} +1 & \text{if} \quad \Lambda^{(T)}_{MAP}(s;\gamma) \geq 1, \\ -1 & \text{otherwise} \end{cases} \tag{4}$$

where

$$\Lambda^{(T)}_{MAP}(s;\gamma) \triangleq \frac{\sum_{v_+ \in \mathcal{V}_+} \sum_{\Delta_+ \in \mathcal{D}_+} exp(-\gamma \|v_+ * \Delta_+ - s\|^2_T)}{\sum_{v_- \in \mathcal{V}_-} \sum_{\Delta_- \in \mathcal{D}_+} exp(-\gamma \|v_- * \Delta_- - s\|^2_T)} \tag{5}$$

and $\mathcal{D}_+ \triangleq 0, ..., \Delta_{max}$.

Importantly, we do not know the latent sources, nor do we know the noise distribution is i.i.d. Gaussian. We assume that we have access to training data as in Weighted Majority Voting section. We make a further assumption that the training data were sampled out of the latent source model and that we have n different training time series. Denote $n_+ \triangleq |\mathcal{R}_+|, n_- \triangleq |\mathcal{R}_-|, \mathcal{R} \triangleq \mathcal{R}_+ \cup \mathcal{R}_-$, and $\mathcal{D} \triangleq \{-\Delta_{max}, ..., 0, ..., \Delta_{max}\}$. Then, we could approximate the MAP classifier by using training data as a proxy for the latent sources. Specifically, replace the inner sum by a minimum in the exponent, replace $\mathcal{V}_+$ and $\mathcal{V}_-$ by $\mathcal{R}_+$ and $\mathcal{R}_-$, and replace $\mathcal{D}_+$ by $\mathcal{D}$ to obtain the ratio:

$$\Lambda^{(T)}(s;\gamma) \triangleq \frac{\sum_{r_+ \in \mathcal{R}_+} exp(-\gamma(\min_{\Delta_+ \in \mathcal{D}} \|r_+ * \Delta_+ - s\|^2_T))}{\sum_{r_- \in \mathcal{R}_-} exp(-\gamma(\min_{\Delta_- \in \mathcal{D}} \|r_- * \Delta_- - s\|^2_T))} \tag{6}$$

Then, it yields the weighted majority voting rule. If we didn't replace the summations over the time shifts with minimums and our main result for weighted majority voting to follow would still hold using the same proof.

In application, we may call for trading off true and false positive rates. One way to do so is to generalize the decision rule to declare the label to be +1 if $\Lambda^{(T)}(s,\gamma) \geq \theta$ and then sweep over different values of parameter $\theta > 0$. The resulting decision rule, which we refer to as generalized weighted majority voting is thus:

$$\hat{L}_{\theta}^{(T)}(s,\gamma) = \begin{cases} +1 & \text{if} \quad \Lambda^{(T)}(s,\gamma) \geq \theta, \\ -1 & \text{otherwise} \end{cases} \tag{7}$$

where setting $\theta = 1$ recovers the usual weighted majority voting.

2.3 Theoretical Guarantee of Misclassification

In Dr.Chen's paper[1], they proposed a performance guarantee for generalized weighted majority voting. First, we define the gap between $\mathcal{R}_+$ and $\mathcal{R}_-$ restricted to time length T and with maximum time shift Δ_{max} as:

$$G^{(T)}(\mathcal{R}_+, \mathcal{R}_-, \Delta_{max}) \triangleq \min_{r_+ \in \mathcal{R}_+, r_- \in \mathcal{R}_-, \Delta \in \mathcal{D}} ||r_+ * \Delta_+ - r_- * \Delta_-||_T^2 \tag{8}$$

This quantity measures how far apart the two different classes are if we only look at length-T chunks of each time series and allow all shifts of at most Δ_{max} time steps in either direction.

Theorem 2.1. *Let $m_+ \triangleq |\mathcal{V}_+|$ be the number of latent sources with label +1, and $m_- \triangleq |\mathcal{V}_-| = m - m_+$ be the number of latent sources with label -1. For any $\beta > 1$, under the latent source model with $n > \beta m log m$ time series in the training data, the probability of misclassification time series S with label L using generalized weighted majority voting $\hat{L}_{\theta}^{(T)}(s,\gamma)$ satisfies the bound.*

$$\begin{aligned} &\mathbb{P}(\hat{L}_{\theta}^{(T)}(S,\gamma) \neq L) \\ &\leq (\frac{\theta m_+}{m} + \frac{m_-}{\theta m})(2\Delta_{max}+1) n exp(-(\gamma - 4\sigma^2\gamma^2) G^{(T)}(\mathcal{R}_+, \mathcal{R}_-, \Delta_{max})) + m^{-\beta+1} \end{aligned} \tag{9}$$

Given error tolerance $\delta \in (0,1)$ and with choice $\gamma \in (0, \frac{1}{4\sigma^2})$, then upper bound is at most δ if $n > m log \frac{2m}{\delta}$, and

$$G^{(T)}(\mathcal{R}_+, \mathcal{R}_-, \Delta_{max}) \geq \frac{log(\frac{\theta m_+}{m} + \frac{m_-}{\theta m}) + log(2\Delta_{max}+1) + log n + log\frac{2}{\delta}}{\gamma - 4\sigma^2\gamma^2} \tag{10}$$

This means that if we have access to a large enough pool of labeled time series, i.e., the pool has $\Omega(m log \frac{m}{\delta})$ time series, then we can subsample $n = \Theta(m log \frac{m}{\delta})$ of them to use as training data. Then with choice $\gamma = \frac{1}{8\sigma^2}$, generalized weighted majority voting correctly classifies a new time series S with probability at least $1 - \delta$ if

$$G^{(T)}(\mathcal{R}_+, \mathcal{R}_-, \Delta_{max}) = \Omega(\sigma^2(log(\frac{\theta m_+}{m} + \frac{m_-}{\theta m}) + log(2\Delta_{max}+1) + log\frac{m}{\delta})) \tag{11}$$

Thus, the gap between sets $\mathcal{R}_+$ and $\mathcal{R}_-$ needs to grow logarithmic in the number of latent sources m in order for weighted majority voting to classify correctly with high probability.

Notation $f(n) = \Omega(g(n))$ is defined as $g(n) = O(f(n))$. $f(n) = \Theta(g(n))$ is defined as $f(n) = O(g(n))$ and $g(n) = O(f(n))$ which is stronger than big O.

Proof. For each time series, there exist a latent source V, shift $\Delta' \in \mathcal{D}$, and noise signal E' such that

$$S = V * \Delta' + E' \tag{12}$$

With a training set of size $n \geq \beta mlogm$, for each latent source $V \in \mathcal{V}$, there exist at least one signal in $\mathcal{R}$ that is generated from V.

Then, we note that R is generated from V as

$$R = V * \Delta'' + E'' \tag{13}$$

where $\Delta = \Delta' - \Delta'' \in D$ and $E = E' - E'' * \Delta$. Since E' and E'' are i.i.d. over time and sub-Gaussian with parameter σ, one can easily verify that E is i.i.d. over time and sub-Gaussian with parameter $\sqrt{2}\sigma$.

Now, we want to bound the probability of error of classifier. The probability of error or misclassification using the first T samples of S is given by

$$\begin{aligned} &\mathbb{P}(\text{missclassification of S using its first T samples}) \\ &= \mathbb{P}(\hat{L}_\theta = -1 | L = +1)\mathbb{P}(L = +1) + \mathbb{P}(\hat{L}_\theta = +1 | L = -1)\mathbb{P}(L = -1) \end{aligned} \tag{14}$$

where we know that $\mathbb{P}(L = +1) = \frac{m_+}{m}$. Then, we will focus on bounding $\mathbb{P}(\hat{L}_\theta = -1 | L = +1)$. By the Markov's inequality,

$$\mathbb{P}(\hat{L}_\theta = -1 | L = +1) = \mathbb{P}(\frac{1}{\Lambda^{(T)}} \geq \frac{1}{\theta} | L = +1) \leqslant \theta \mathbb{E}[\frac{1}{\Lambda^{(T)}} | L = +1] \tag{15}$$

Now,

$$\mathbb{E}[\frac{1}{\Lambda^{(T)}} | L = +1] \leqslant \max_{r_+ \in \mathcal{R}_+, \Delta_+ \in \mathcal{D}} \mathbb{E}[\frac{1}{\Lambda^{(T)}(r_+ * \Delta_+ + E; \gamma)}] \tag{16}$$

Let $r_+ \in \mathcal{R}$ and $\Delta_+ \in \mathcal{D}$. Then for any time series s,

$$\Lambda^{(T)}(s; \gamma) \geq \frac{exp(-\gamma ||r_+ * \Delta_+ - s||_T^2)}{\sum_{r_- \in \mathcal{R}_-, \Delta_- \in \mathcal{D}} exp(-\gamma ||r_- * \Delta_- - s||_T^2)} \tag{17}$$

After evaluating the above for $s = r_+ * \Delta_+ + E$, we could see that

$$\begin{aligned} &\frac{1}{\Lambda^{(T)}(r_+ * \Delta_+ + E; \gamma)} \\ &\leq \sum_{r_- \in \mathcal{R}, \Delta_- \in \mathcal{D}} \{exp(-\gamma ||r_+ * \Delta_+ - r_- * \Delta_-||_T^2) exp(-2\gamma \langle r_+ * \Delta_+ - r_- * \Delta_-, E \rangle_T)\} \end{aligned} \tag{18}$$

where we denote $\langle q, q' \rangle_T \triangleq \sum_{t=1}^{T} q(t)q'(t)$, and $||q||_T^2 \triangleq \langle q, q \rangle_T$. Then, we could get the following bound:

$$\begin{aligned}
&\mathbb{E}\left[\frac{1}{\Lambda^{(T)}(r_+ * \Delta_+ + E; \gamma)}\right] \\
&\leq \mathbb{E}\left[\sum_{r_- \in \mathcal{R}, \Delta_- \in \mathcal{D}} \{exp(-\gamma||r_+ * \Delta_+ - r_- * \Delta_-||_T^2) exp(-2\gamma \langle r_+ * \Delta_+ - r_- * \Delta_-, E \rangle_T)\}\right] \\
&= \sum_{r_- \in \mathcal{R}_-, \Delta_- \in \mathcal{D}} exp(-\gamma||r_+ * \Delta_+ - r_- * \Delta_-||_T^2) \prod_{t=1}^{T} \mathbb{E}\left[exp(-2\gamma(r_+(t+\Delta_+) - r_-(t+\Delta_-))E(t))\right] \\
&\leq \sum_{r_- \in \mathcal{R}_-, \Delta_- \in \mathcal{D}} exp(-\gamma||r_+ * \Delta_+ - r_- * \Delta_-||_T^2) \prod_{t=1}^{T} exp(4\sigma^2\gamma^2(r_+(t+\Delta_+) - r_-(t+\Delta_-))^2) \\
&= \sum_{r_- \in \mathcal{R}_-, \Delta_- \in \mathcal{D}} exp(-(\gamma - 4\sigma^2\gamma^2)||r_+ * \Delta_+ - r_- * \Delta_-||_T^2) \\
&\leq (2\Delta_{max} + 1) n_- exp(-(\gamma - 4\sigma^2\gamma^2) G^{(T)})
\end{aligned} \tag{19}$$

where we use the independence of entries of E and we assume that E is zero mean Gaussian distribution.

Now, we obtain that

$$\mathbb{P}(\hat{L}_\theta(s; \gamma) = -1 | L = +1) \leq \theta(2\Delta_{max} + 1) n_- exp(-(\gamma - 4\sigma^2\gamma^2) G^{(T)}) \tag{20}$$

Repeating a similar argument yields

$$\mathbb{P}(\hat{L}_\theta(s; \gamma) = +1 | L = -1) \leq \frac{1}{\theta}(2\Delta_{max} + 1) n_+ exp(-(\gamma - 4\sigma^2\gamma^2) G^{(T)}) \tag{21}$$

Finally, we obtain the bound as

$$\begin{aligned}
&\mathbb{P}(\text{misclassification of S using its first T samples}) \\
&\leq \theta(2\Delta_{max} + 1)\frac{n_- m_+}{m} exp(-(\gamma - 4\sigma^2\gamma^2) G^{(T)}) + \frac{1}{\theta}(2\Delta_{max} + 1)\frac{n_+ m_-}{m} exp(-(\gamma - 4\sigma^2\gamma^2) G^{(T)}) \\
&= \left(\frac{\theta m_+}{m} + \frac{m_-}{m\theta}\right)(2\Delta_{max} + 1) n exp(-(\gamma - 4\sigma^2\gamma^2) G^{(T)})
\end{aligned} \tag{22}$$

□

3 Application: Twitter Trending Topics Prediction

In this section, we consider the application of the method in section 2 toward detection of trending topics on Twitter. We discuss the Twitter service, the collection and preprocessing of data, and the experimental setup for the detection task.

3.1 Overview

3.1.1 Overview of Twitter

Overview of the twitter service is from Twitter official website, papers, and wikipedia. Twitter is a real-time messaging service and information network. Users of Twitter can post short (up to 140 characters) messages called *Tweets*, which are then broadcast to the users *followers*. Users can also engage in conversation with one another. By default, Tweets are public, which means that anyone can see them and potentially join a conversation on a variety of topics being discussed. Inevitably, some topics gain relatively sudden popularity on Twitter. For example, a popular topic might reflect an external event such as a breaking news story or an internally generated inside joke or game. Twitter surfaces such topics in the service as a list of top ten trending topics.

3.1.2 Twitter-Related Definitions

Talking about Tweets, topics, trends and trending topics can be ambiguous, so here we make precise our usage of these and related terms.

Definition 3.1. *We define a* ***topic*** *to be a phrase consisting of one or more words delimited by spacing or punctuation. A word may be any sequence of characters and need not be an actual dictionary word.*

Definition 3.2. *A Tweet is* ***about*** *a topic if it contains the topic as a substring. Tweets can be about many topics.*

Example 3.1. *The following tweet by the author (handle @Zsquared) contains the string "AMSC663" Hence, it is considered to be about the topic "AMSC663".*

"AMSC663 Project drives me crazy."

Definition 3.3. *A* ***trending topic*** *is a topic that is currently on the list of trending topics on Twitter. If a topic was ever a trending topic during a period of time, we say that the topic* ***trended*** *during that time period.*

Definition 3.4. *A trending topic will also occasionally be referred to as a* ***trend*** *for short.*

Definition 3.5. *The* ***trend onset*** *is the time that a topic first trended during a period of time.*

Example 3.2. *If the topic "government shutdown" is currently in the trending topics list on Twitter, we say that "government shutdown" is trending, and that is a trend. The topic "government shutdown" has a trend onset, which is the first time it was trending in a given period of time. This could, for example, correspond to when the "shutdown" happened. After "government" is no longer trending, we say that "government shutdown" trended.*

3.1.3 Problem Statement

At any given time there are many topics being talked about on Twitter. Of these, some will trend at some point in the future and others will not. We wish to predict which topics will trend. The earlier we can predict that a topic will trend, the better. Ideally, we would like to do this while maintaining a low rate of error.

3.1.4 Proposed Solution

Our approach to detecting trending topics is as follows. First, we gather examples of topics that trended and topics that did not trend during some period of time. Then, for each topic, we collect Tweets about that topic and generate a time series of the activity of that topic over time. We then use those time series as reference signals and apply the classification method we discussed in previous sections.

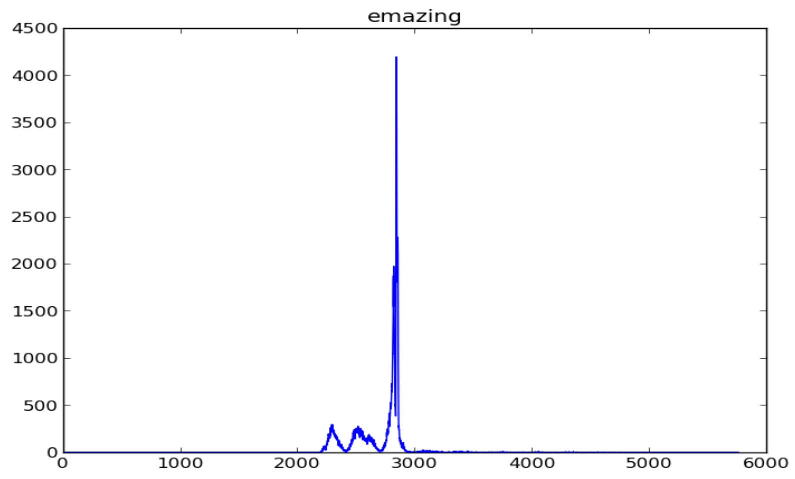

Figure 1: Trending topic Emazing

3.2 Data

3.2.1 Data Collection

The classification model requires a set of reference signals corresponding to topics that trended and a set of reference signals corresponding to topics that did not trend during a time window of interest. These reference signals represent historical data against which we can compare our most recent observations to do classification.

The data collection process can be summarized as follows. First, we collected 200 examples of topics that trended at least once and 200 examples of topics that never trended on November 2013. We then sampled Tweets from this sample window and labeled each Tweet according to the topics mentioned therein. Finally, we constructed a reference signal for each topic based on the Tweet activity corresponding to that topic.

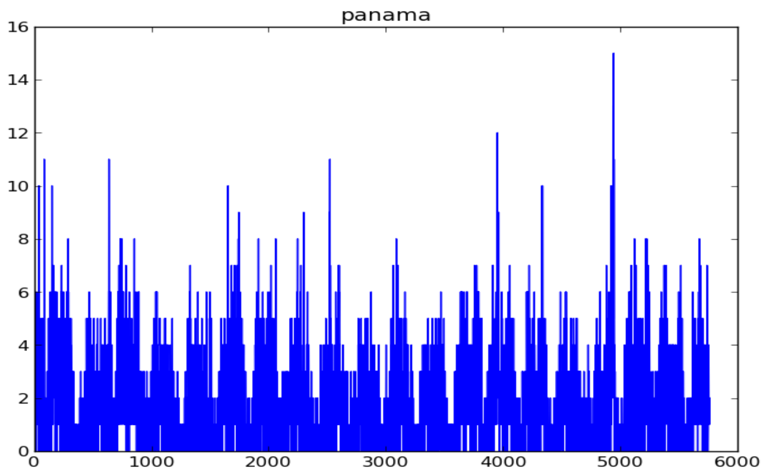

Figure 2: Non-Trending Topic panama

Proper Topic Signal Filtration

We collected a list of all trending topics on Twitter as well as the trending times and their rank in the trending topics list on Twitter. We filtered out topics whose rank was never better than or equal to 3. In addition, we filtered out topics that did not trend for more than 30 mins. One topic cannot be trending multiple times in one day. Such trending signal is shown in Figure 1.

For topics that were not trending, we first sampled a list of n-grams (phrases consisting of n "words") occurring on Twitter during the sample window for n up to 5. We filtered out n-grams that contain any topic that trended during the sample window, using the original, unfiltered list of all topics that trended during the sample window. Such non-trending signal is shown in Figure 2.

3.2.2 From Tweets to Signals

Per topic, we creates its time series based on a pre-processed version of the raw rate of how often the topic was shared,i.e.,its Tweet Rate. We empirically found that how news topic become trends tends to follow a finite number of patterns; a few examples of these patterns are shown in Figure 3. However, before processing the signal, it is obvious we need to normalize the signals to do the comparison. In [3], Several normalization technique is proved to be efficient for time series with the similar feature as twitter signals. We could see the normalized signal in Figure 4. The trending and nontrending signal is easily classified even with eyes.

Baseline Normalization

Many non-trending topics have a relatively high rate and volume of Tweets, and many trending

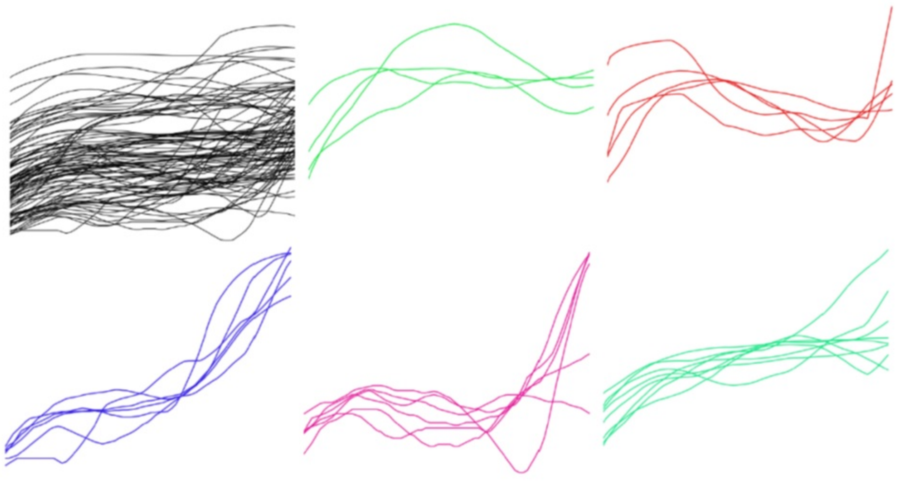

Figure 3: How topics become trends on Twitter. If we seperate the cluster, we could find finite patterns that a topic signal will trend

topics have a relatively low rate and volume of Tweets. One important difference is that many non-trending topics have a high baseline rate of activity while most trending topics are preceded by little. For example, a non-trending topic such as “New York” is likely to have a consistent baseline of activity because it is a common word. To emphasize the parts of the rate signal above the baseline and de-emphasize the parts below the baseline, we define a baseline b as

$$b = \sum_t \rho[t] \tag{23}$$

and a baseline normalized signal ρ_b as

$$\rho_b[t] = (\frac{\rho[t]}{b})^\beta \tag{24}$$

The exponent β controls how much we reward and penalize rates above and below the baseline rate. We use $\beta = 1$ in this project.

Spike Normalization

Another difference between the rates of Tweets for trending topics and that of nontrending topics is the number and magnitude of spikes. We emphsize such spikes, while de-emphasizing smaller spikes.

$$\rho_{b,s}[t] = |\rho_b[t] - \rho_b[t-1]|^\alpha \tag{25}$$

in terms of the already baseline-normalized rate ρ_b. The parameter $\alpha > 1$ controls how much spikes are rewarded. We use $\alpha = 1.2$.

Smooth

Tweet rates, and the aforementioned transformations thereof, tend to be noisy, especially for small time bins. We convolve the signal with a smoothing window of size N_{smooth}. Applied to the spike-and-baseline-normalized signal $\rho_{b,s}$, this yields the convolved version

$$\rho_{b,s,c}[t] = \sum_{m=t-N_{smooth}+1}^{t} \rho_{b,s}[m] \tag{26}$$

Branching Processes and Logarithmic Scale

It is reasonable to think of the spread of a topic from person to person as a branching process. It is also reasonable to measure the volume of tweets at a logarithmic scale to reveal these details. Here we may have problems if the signal vector have zero entries. If the entries are zero, we will set them as zero too in this Scaling step.

$$\rho_{b,s,c,l}[t] = \log(1 + \rho_{b,s,c}[t]) \tag{27}$$

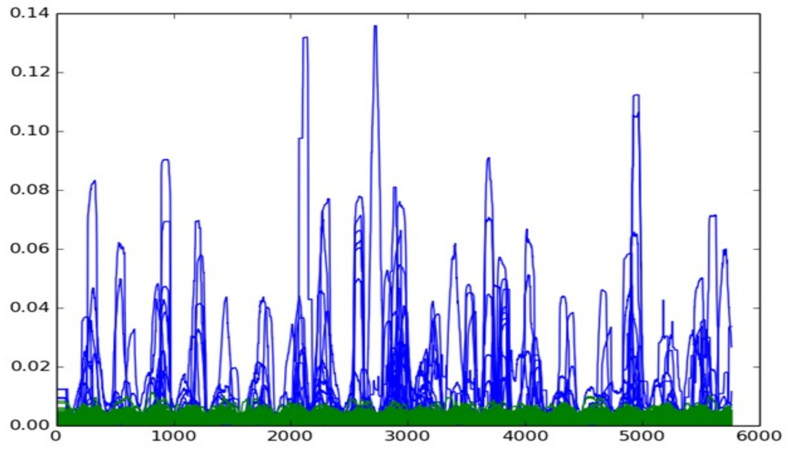

Figure 4: Normalized Trending signal (blue) vs. Normalized nontrending signal (green)

3.3 Experiment

We divide the set of topics into a training set and a test set using a 50/50 split. For each topic in the test set, we wish to predict if the topic will trend. If the topic really did trend, we wish to detect it as early as possible relative to the true trend onset while incurring minimal error.

3.3.1 Detection Setup

In principle, to test the detection algorithm, one would step through the signal in the entire sample window for each topic in the test set and report the time of the first detection. In practice, we take a shortcut to avoid looking through the entire signal based on the following observations about the activity of topics that trended and topics that did not.

First, for topics that trended, there is little activity aside from that surrounding the true onset of the trend. In the rare event that a detection is made very far from the true onset, it is reasonable to assume that this corresponds to a completely different event involving that topic and we can safely ignore it. Thus, the only part of the signal worth looking at is the signal within some time window from the true onset of the trend.

Second, topics that did not trend exhibit relatively stationary activity. Therefore, it is reasonable to perform detection only on a piece of the signal as an approximation to the true detection performance.

Definition 3.6. *Let h_{ref} be the number of hours corresponding to N_{ref} samples. At 2 minutes per sample, h_{ref} is given by $N_{ref}/30$.*

For TEST topic signals that have trended, we do detection on the window spanning $2h_{ref}$ hours centered at the true trend onset. For topics that did not trend, we randomly choose a window of the desired size.

3.3.2 Parameter Exploration and Trials

We explore all combinations of the following ranges of parameters, excluding parameter settings that are incompatible (e.g. $N_{obs} > N_{ref}$). For each combination, we conducted 5 random trials.

- γ: 0.1, 0.5, 1.0, 5.0, 10.0
- N_{obs}: 10.0, 30.0, 50.0, 100.0, 120.0
- N_{smooth}: 20.0, 40.0, 80.0, 120.0, 160.0, 300.0
- h_{ref}: 4, 6, 10, 14, 17
- D_{req}: 1, 2, 3, 4
- θ: 0.65, 1, 3

Notation N_{obs} corresponds the number of time bins. However, in the codes or other place, we may use M_{obs} with unit minute. The size of the time bin is 5 mins in the experiment. Given user may want to change the size of time bin, we use M_{obs} instead of N_{obs} for convenience in the software. h_{ref} is with unit hour. H_{ref} corespond the number of time bins of reference signal. N_{smooth} is with unite time bin.

3.3.3 Evaluation

To evaluate the performance of our method, we compute the false positive rate and true positive rate for each experiment, averaged over all trials. In the case of true detections, we compute the detection time relative to the true onset of the trending topic.

Before presenting the results, we need to note that twitter produces a list of top ten trending topics, while we performs detection based on a score and threshold, and do not limite the number of topics detected as trending at any given time. For example, a topic may be detected as trending by our model, however, since there may be some more popular topics, twitter will not add such topic on the trending list. Despite this limitation, our model and implementation work well.

3.4 Results and Discussion

In this section, we present the results of the trend detection experiment described in last section. We show the quality of the trend detection algorithm using ROC curves and distributions of detection time relative to the true trend onset. We analyze: 1) The effect of FPR and TPR on Relative Detection Time; 2) The effect of Parameters on Position Along ROC Curve; 3) Effect of Parameters on Movement Along ROC Curve.

3.4.1 The effect of FPR and TPR on Relative Detection Time

In the experiment, we run 5 trials for each parameter set. We have total 9000 parameter set for this experiment and the experiemnt generates the distribution of parameter set on ROC space. In Figure 6, we can see different parameter set result in the tradeoff between TPR and FPR. To simplify our analysis, we break it into three regions: the top region, the center region, and the bottom region. More precisely, we define the regions as follows:

Definition 3.7. *(Top Region) (FPR, TPR) is in the top region if* $FPR > 0.25$ *and* $TPR > 0.75$.

Definition 3.8. *(Center Region) (FPR, TPR) is in the center region if* $FPR \leq 0.25$ *and* $TPR > 0.75$.

Definition 3.9. *(Bottom Region) (FPR,TPR) is in the bottom region if* $FPR \leq 0.25$ *and* $TPR \leq 0.75$.

In the top region, we accept the possibility of frequent false detections for the sake of rarely missing the chance to make a true detection. In the bottom region, we accept a lower chance of making a true detection for the sake of rarely making false detections. The center region lies in between these two extremes. Consequently, points in the top region correspond to earlier detection relative to the true onset of a trend as detected by Twitter, points in the bottom, correspond to predominantly late detection, and points in the center roughly balance being early and late. Figure 5-8 illustrates this.

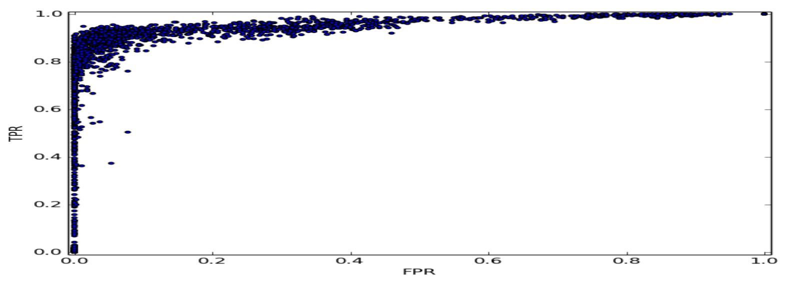

Figure 5: Parameter set distribution on ROC space

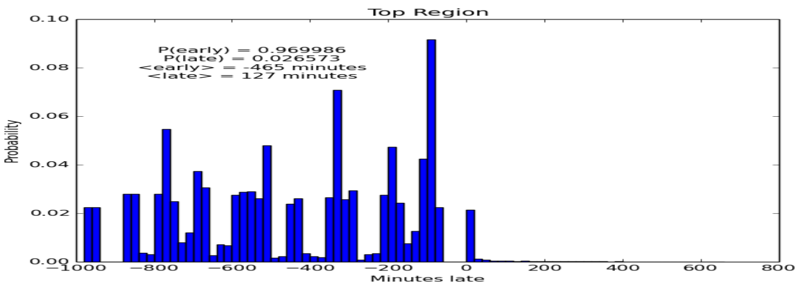

Figure 6: Relative Detection Time for parameter sets in top region

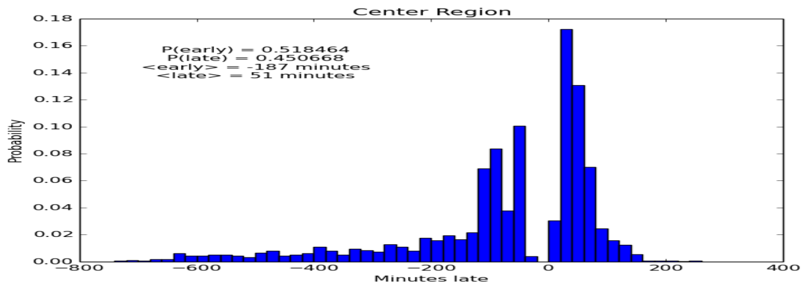

Figure 7: Relative Detection Time for parameter sets in center region

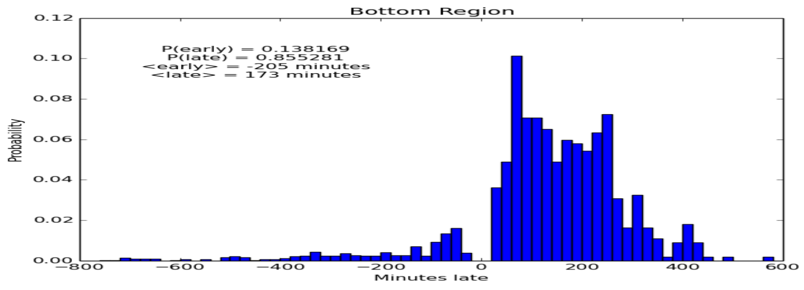

Figure 8: Relative Detection Time for parameter sets in bottom region

Table 1: The effect of parameter on position along the ROC curve

	$< M_{obs} >$	$< h_{ref} >$	$< \gamma >$	$< D_{req} >$	$< \theta >$	$< N_{smooth} >$
top	192.31	11.59	4.82	2.49	0.73	108
center	220.73	10.45	3.38	2.52	1.00	108
bottom	214.89	11.07	1.45	3.61	2.72	118

3.4.2 Effect on Position of ROC space

In this section, we analyze the effect of each parameter on the position of ROC curve. To simplify analysis, we again consider only the top, center and bottom regions. In table one, we could see that θ is dramatically different in each region. Intuitively, larger θ will make more negative decisions and our results coincide it. Besides, we could see that γ and D_{req} are both very different in each region. Our results coincide again the theoretical explanation. Larger D_{req} will prevent the classifier from making positive decisions so that D_{req} is larger in bottom region. As in section 2, we know γ defines the scaling of parameters "‘sphere"’ of influence of each reference signal. Larger γ results in relatively larger class ratio, and the ratio is then easily greater than θ. M_{obs}, h_{ref}, and N_{smooth} barely changes and has no significant effect on position along the curve.

3.4.3 Effect on Movement Along ROC Curve

In this section, we have a parameter that varies to produce the ROC curve while other parameters are fixed. We show how varying a given parameter trades off TPR for FPR by computing the discrete derivative of FPR and TPR with respect to the parameter. For example, we use p to denote the parameter variable and compute:

$$\Delta_{p,i}^{FPR} = \frac{FPR(p_i) - FPR(p_{i-1})}{p_i - p_{i-1}}$$

$$\Delta_{p,i}^{TPR} = \frac{TPR(p_i) - TPR(p_{i-1})}{p_i - p_{i-1}}$$

for each ROC curve associated with p and for i ranging from the second to the last in increasing order. By using the parameter optimization codes, we actually use 5 trials to compute one point on the ROC curve. Then, we compute the above respect to the variable parameter over all possible fixed combination and then find the average $< \Delta_p^{FPR} >$ and $< \Delta_p^{TPR} >$.

The results is a distribution of discrete derivatives of FPR and TPR with respect to a variable parameter p which tells us the tradeoff between TPR and FPR. If $< \Delta_p^{FPR} >$ and $< \Delta_p^{TPR} >$ are greater than zero, then an increase in p causes a movement toward to (1,1) on ROC space, and vice versa. Sometimes, the curve moces neither toward to (0,0) nor toward to (1,1) but toward to (0,1) or (1,0). The former represents an increase in TPR in addition to a decrease in FPR – a win-win

Table 2: $< \Delta_p^{FPR} >$

	M_{obs}	h_{ref}	γ	D_{req}	θ	N_{smooth}
top	-0.0013	0.0224	0.1096	-0.0436	-0.9400	0.0020
center	-0.0045	0.0134	0.058	-0.0118	-0.0546	0.0001
bottom	0.0002	0.0201	0.0002	-0.0021	-0.0002	0.0000

Table 3: $< \Delta_p^{TPR} >$

	M_{obs}	h_{ref}	γ	D_{req}	θ	N_{smooth}
top	-0.0002	0.0017	0.0079	-0.0147	-0.1900	0.0000
center	0.0002	0.0001	0.0265	-0.0225	-0.0118	0.0002
bottom	0.0011	0.0092	0.3558	-0.0531	-0.1050	0.0021

situation. The latter one is the worst situation.

Since the movement along the curve caused by changes in each parameter and the behavior is not uniform, we still consider the average changes in different regions. It can also help us to choose parameter sets given considering the tradeoff between TPR, FPR and Prediction Time.

In table 2 and 3, we can see the movement along the curve caused by changes in each parameter. It is not surprising that θ, which has the most influence on the position along the curve, also has by far the most influence on the movement along the curve. Similarly, a larger D_{req} always moves us down the curve, also expected.

The γ also has big influence. It always moves up the curve. If we increase M_{obs}, then it will move down the curve given starting in the top region and move up the curve given starting in other regions. The length of the reference signals in hours moves down the curve given starting in top and move up the curve given starting in bottom region.

As we know that the parameters θ and γ has significant influence on position along ROC curve. To better understand the effect of θ and γ, Dr.Balan suggests to compute the Δ as following:

$$\Delta_{p_i}^{FPR} = \frac{FPR(p_i) - FPR(p_i - \delta)}{\delta}$$

where $\delta = 0.1$ for instance. Similarly, we compute $\Delta_{p_i}^{TPR}$ as same way.

In order to find the optimal parameter, we actually need to first choose which region we want because of the trade off between detection time and FPR,TPR. It is the reason we do such test to analyze the effect in different regions. Dr.Balan's suggestion is more informative, and we will investigate the effect in the future by applying Dr.Balan's suggestion in the future.

3.4.4 Recommended Parameter Settings

The parameter optimization need to user to input the weight of type 1 error and type 2 error. The the parameter optimization will minimize the cost function $cost = weight_{type1} * error_{type1} + weight_{type2} * error_{type2}$. As we see, different regions in the ROC space will give us different prediction time. Hence, user need to choose a region first and then input the weight, and user may vary some parameters according to the sensitivity of each parameters.

Cost(FP)<<Cost(TP)
We recommend the parameter settings in the third row of Table 1, corresponding to the bottom region. For finer tunning, we can increase h_{ref}, and decrease γ, and increase \$D$_{req}$.

Cost(FP)>>Cost(TP)
We recommend the parameter settings in the first row of Table 1, corresponding to the top region. For finer tunning, we can decrease h_{ref}, γ to increase TP and decrease FP.

Cost(FP)≈Cost(TP)
We recommend the parameter in the second row of Table 1, corresponding to the center region. For finer tunning, we can increase N_{obs} to increase TP and decrease FP simultaneously.

4 Implementation

In this section, we discuss the programming strategy to implement the mathematical model to predict the trending topics. The software package including three parts: 1) Data Preprocessing 2) Parameter Optimization 3)Streaming Predictor.

4.1 Core Detection Algorithm

In this section, we summarize the core detection algorithm used in the parameter optimization and predictor parts. This algorithm contains the core detection logic. In the algorithm, we apply some distance computation function which is generated from the Latent Source Model Decision Rule. At each time step, it updates the observation s so that s contains the latest N_{obs} samples from the infinite stream and computes the distances to positive and negative reference signals. A detection is declared whenever the ratio of class probability $R(s)$ exceeds the threshold θ for D_{req} consecutive time steps. We will discuss the performance and scalability of this algorithm in section 4.5.

Algorithm 1 Perform oneline binary classification on the infinite stream s_∞ using sets of positive and negative reference signals R_+ and R_-.

```
procedure DETECT($s_\infty$, $R_+$, $R_-$, $\gamma$, $\theta$, $D_{req}$)
    $ConsecutiveDetections \leftarrow 0$
    loop
        s $\leftarrow$ $UpdateObservation(s_\infty, N_{obs})$
        for r in $R_+$ do
            $PosDists.APPEND(DISTTOREFERENCE(s, r))$
        end for
        for r in $R_-$ do
            $NegDists.APPEND(DISTTOREFFERENCE(s, r))$
        end for
        $R = PROBCLASS(PosDists, \gamma)/PROBCLASS(NegDists, \gamma)$
        if $R > \theta$ then
            if $ConsecutiveDetections > D_{req}$ then
                $DetectionTime \leftarrow CURRENTTIME()$
                return $DetectionTime$
            else
                $ConsecutiveDetections \leftarrow ConsecutiveDetections + 1$
            end if
        else
            $ConsecutiveDetections \leftarrow 0$
        end if
    end loop
```

4.2 Data Preprocessing

4.2.1 Overview

The raw Twitter data is formatted in JSON and is around 600MB per day. The big size makes the data preprocessing very time consuming because we have I/O bound for a single computer. We proposed two solutions:

- Apply distributed computing algorithm. The data are divided and stored in different clusters, so we can read the whole data parallely. We apply MapReduce programming model to design our own MapReduce codes.
- Since the twitter raw data is enriched(well organized), we inspect the format manually and consider each piece of twitter data as a string without robust JSON parser which is slow.

4.2.2 MapReduce

MapReduce is a programming model for processing large data sets with a parallel, distributed algorithm on clusters. Our MapReduce is implemented by Python. The *Map()* is a procedure that performs filtering and sorting. In our case, it uses robust JSON parser to read all JSON data and generate and sorted "hashtag:Time:Count" . The *Reduce()* is a procedure that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). In our case, it constructs the hashtag signal. It put the count into every time bin. The "MapReduce System" (also called "infrastructure" or "framework") orchestrates by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance.

- Advantages: Read big data parallely and Construct the topic signal very fast
- Disadvantanges: We do need distributed systems and cannot run it on our own computer.

4.2.3 Non-Distributed MapReduce

Since MapReduce need the infrastructure Hadoop, we design and implement a Non-Distributed MapReduce in C. It can contruct the topic signal very fast by only using my own laptop. We chose to implement *Map()* without a fully robust JSON parser because it is very slow. We were able to inspect the format of the JSON manually and determine that we can parse it as a string which is tens of times faster. The output is of the form: $< hashtag >, < timevalue >$.

Reduce() will reduce the sorted mapped data in the form $< hashtag >, < time >$ by granulating the tweet times into $< interval >$ (in minutes) sized bins between $< start >$ and $< end >$ and outputting the results into $< hashtag >$ named files in $< dir\_name >$ directory where each line contains a bin count.

First strategy: we choose to read the whole file generated by *Map()* into the memory and then construct a job queue mastered by one thread. A job is a struct contains $< hashtag >$ and $< pointer \quad array >$. The pointer array contains all pointer pointing to the different time information of the hashtag.The master thread will pass 1000 jobs into each thread, and once the thread finish computing, it will output the hashtag file.

Second strategy: since the data is a stream of sorted hashtags and subsorted time values. We can parse the stream without reading it all into memory, which is not efficient. We only have to allocate enough memory to store the values of one hashtag before dumping it to a file. It proves that the second strategy is faster than first strategy at least for one moneth twitter raw data.

4.2.4 Nature Language Filtering

In spite of MapReduce, we also write a short script python codes to convert unicode encoded strings into actual unicode and then to ASCII equivalent, because our raw twitter data is from South America. The script use the unicode module to find ASCII equivalents of unicode characters and lowercase so that the final result is in pure ASCII. This is beneficial because it reduces the number of unique hashtags solely based off accented characters and such.

4.3 Parameter Optimization

As we see the experiment section, we brute force all the possible parameter sets given users input the parameters. This program takes advantage of multithreading. There were a few options we had on how to use multithreading. We chose to make each parameter set be multithreaded. Each thread gets a fair share of test signals to compute against reference signals and the main threads joins them and combines the results. This was optimal because it is not too little, but not too much work for each thread. Too much work per thread would result in the senario that one thread has to wait for other thread. Too little work would result in much overhead. The parameter optimization record the probability of early detection, expected early detection time, expected late detection time for each parameter set.

4.4 Streaming Predictor

User input the weight of errors and then predictor will choose the optimal parameter from the parameter file generated by parameter optimization part. Predictor takes as input an incoming signal file and compares against the reference signals using the core detection algorithm. It outputs the detect time if detected and either TP, FP, TN, or FN.

We also commented an important part in the codes. In last paragraph, the predictor actually works for classification, not prediction. The commented part simulate the prediction process. We truncate the whole test signal as current signal and historical signal, so the predictor can start to detect at the current state and normalize the current state by using historical signal.

4.5 Utilities

The most important utility function is *arraylist.h*. It contains the detection functions and normalization functions. *arraylist.h* uses X Macroing to easily create different arraylist types. All of these utility functions are $\#define'd$ which is basically forcing them as inline. Since the compiler ultimately decides which functions to inline anyway, we can only guareentee inline by abusing the preprocessor. This gives us performace increase because we reduce (just about all) the time on function call overhead. Our efficient use of the preprocessor in conjuction with the $-Ofast$ and $-ffast-math$ compiler flags speed up our execution time by well over a factor of 10 at the cost of barely any extra memory usage and only about an extra second of compile time. We choose to use an arraylist as the underlying data structure representing the signals because we only need to append to, copy, and index into the signal and implementations of arraylists are generally efficient at that. This was a better choice than a hashtable, and tree because we do not ever search the signal for a specific value, which would be $O(n)$ for an unsorted arraylist. All of the operations we do are $O(1)$ (constant time) for the arraylist implementation, so it's the obvious choice of data structure to represent the signals.

4.6 Validation

For preprocessing part, we have two packages to validate each other. The distributed MapReduce algorithm uses robust JSON parser to generate the signals and the non distributed C algorithm read JSON as string to generate the signals. We could generate exactly the same signals. The preprocessing is validated.

For parameter optimization and predictor, the first validation strategy is to use the python codes I wrote to run the same data sets and compare whether they can generate the same results. The second validation strategy is to implement the equation(5) instead of equation(6) to validate the codes. Our codes only use minimum distance as your see in equation(6), but the theoretical model is supposed to sum all the distance together between test signal and all possible reference signals. The genereate the same results so that the codes are validated. The third strategy is to input synthetic signals print out some intermedian results on screen and compare them with the validation python codes. Therefore, the whole software is validated by these strategies.

In the codes, we also include many error condition to remind us if we are running the wrong data set or wrong parameter sets.

4.7 Complexity Analysis

To understand how well our implementation is, we analyze the complexity of our parameter optimization codes and predictor codes. Our core detection algorithm is shown in algorithm 1. Therefore, most of our theoretical analysis depends on the algorithm.

4.7.1 Background

In the project, the topic signal has one month length. Given the time bin is 5mins, we can easily see the signal has around 8640 time bins. In this project, we collect 200 trending signals and 200 non trending signals. As we mentioned, all reference signal have same length depending on h_{ref}, where unit of h_{ref} is in hours. The length of the test signal is fixed in the parameter optimization as $2h_{ref}$, and is fixed as one month length in predictor. We make a decision every time when we compute the distance between one piece of test signal with length M_{obs}, unit in minutes, and all possible pieces of all reference signals. If we could have a consecutive D_{req} positive decision, then we stop the detection procedure. Therefore, the worst case is we never satisfy the stop critier and finally give a negative decision. Given the worst case, per decision making, we need to do $N_{obs}(h_{ref} - N_{obs})(\|R_+\| + \|R_-\|)$ comparisons between elements of the test signal and elements of reference signals.

4.7.2 Parameter Optimization

Because we could run the parameter optimization offline, we don't do much complexity analysis on this part. We still apply parallellization on programming parameter optimization. To understand the parallellization senario, we make a simple example here and explain how we test the real complexity.

First, we randomly split the 200 trending topic into 100 test signals and 100 traning signals. Similarly, we have 100 non trending training signals and 100 test non trending signals. Second, if we have 5 slave threads, the master thread will pass 20 test signals to each thread. Finally, once the master thread get a parameter set, each thread will do the computation and make 20 decisions for the 20 test signal. The slave threads will pass the results back to master thread and master thread combine the results to output True Positive Rate and False Positive Rate for this parameter set. This procedure is called one trial. For each parameter set, user could input the number of trials. In my test, we have 5 random trials for each parameter set, which means we randomly split the 200 signals 5 times and repeat the procedure above every time.

In this senario, the total time bin comparisions for one thread is equal to $20 * N_{obs} * (h_{ref} - N_{obs} + 1) * 100 * 5$. Since our parameter optimization brute force all possible combinations of these 6 parameters(we have 9000 parameter sets in the test), we just increase the number of parameter sets and timing the program. As we expected in Figure 9, time linearly grows. For example, when we have only one parameter set, we measure the time. Then, when we have 2 parameter sets, which means the program will brute force 2 parameter sets, we measure the time.

Intuitively, if we increase the number of test signals, then our TPR and FPR would be more accurate and the time won't grow linearly because of the parallellization. Parallellization speed up the decision making for each parameter, so it speed up the whole program when it have to brute force thousands of parameter sets.

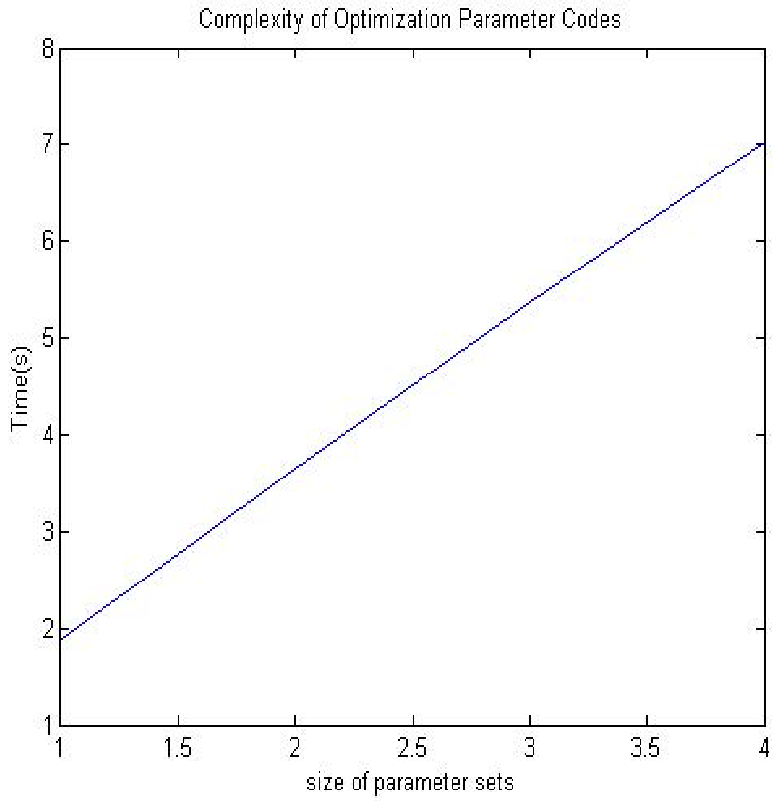

Figure 9: Complexity of Parameter Optimization

4.7.3 Predictor

Predictor will test the online stream in twitter topic signals, so it's complexity is very important. Fortunately, our predictor can make a decision by using 30 seconds in the worst case. The predictor is programmed according to the core detection algorithm 1. However, to reduce the number of shifts in equation (6), we only makes updates from one time bin to another, instead of computing all possible combiantions of distances in equation (6) at every time bin. If we implement latter methods, we have to do $N_{obs}(N_{ref} - N_{obs} + 1)$ shifts while the former method only need $(N_{ref} - N_{obs} + 1)$ shifts. Next, we would explain how we test the predictor complexity.

First, we can analyze the complexity of the detection theoretically. Given the core detection algorithm, the number of comparisions(operations) is $N_{obs}(h_{ref} - N_{obs} + 1)$ if we cover all possible shifts. Then, we make some tests and the results corresponds to the theoretical analysis. In Figure 10, we fixed the h_{ref}, and changes $N_{obs} = M_{obs}/5$. We can see that the worst case is when $M_{obs} = \frac{1}{2}h_{ref}$. In Figure 11, we fixed the $M_{obs} = 250$ and changes the h_{ref}. It is linear as expected.

Also, Dr.Balan suggests another way to speed up the distance computation between two signals. In the algorithm 1, we will update the test signal if we didn't make a positive decision. Every time we update the test signal chuck, my implementation will compute all distances from the current chunk in test signal to all possible chunks of reference signals. Computationally, every time we update the test signal, we need to do $O(N_{obs}(N_{ref} - N_{obs} + 1)(\|R_+\| + \|R_-\|))$ computations. Instead, we could first time compute all distances from current chuck on test signal to chunks of reference signals. Then, we update those distances with the boundary terms every time we update the test signal. Computationall, we only need to do $O(N_{obs} + (N_{obs} - 1))(\|R_+\| + \|R_-\|))$ computations. This method indicates larger N_{obs} will save more computataions. However, this method need more memory. For each reference signal, we need allocate $(N_{ref} - N_{obs} + 1)$ arrays with size N_{obs}. Larger test signal chunk will lead to more memory allocation and access time.

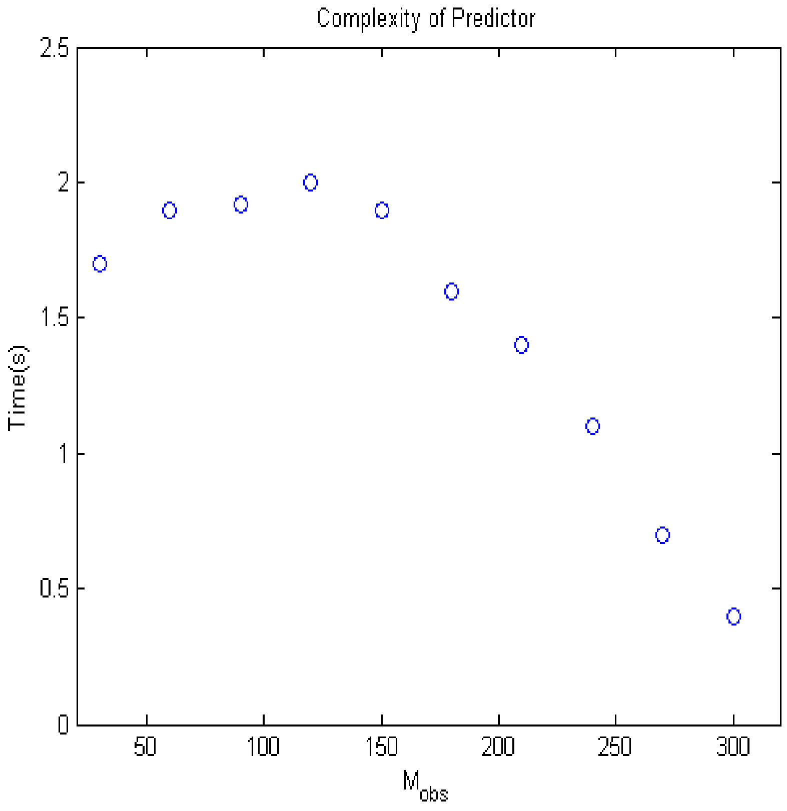

Figure 10: Complexity of Predictor(fixed $h_{ref} = 5h$

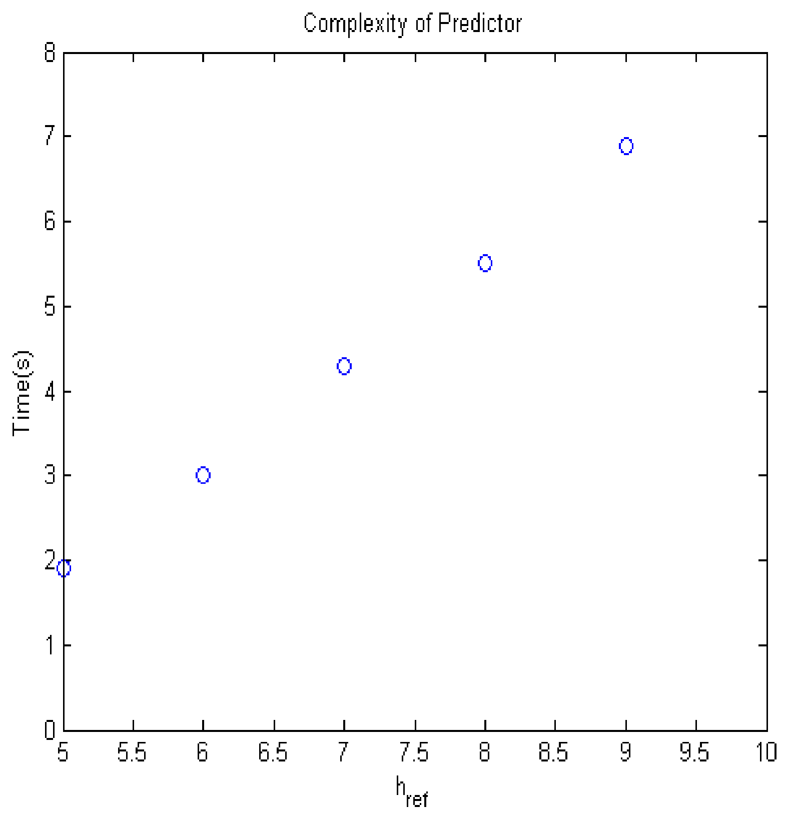

Figure 11: Complexity of Predictor(fixed $M_{obs} = 250mins$

5 Further Work

Our final goal is to use this model to predict a protest on street. In the streaming predictor, we add another threshold regarding to the number of tweets around the detected spike. It can prevent from detecting some local spike which does not correspond to the event. We need to make a good estimate of the threshold and update the estimate overtime, hence, the model can predict some protests.

Computationally, we could use a more sophisticated algorithm instead of our core detection algorithm. For instance, Rakthanmanono have shown a way to efficiently search over trillions of time series subsequences. Since our probability based metric involves exponential decay based on the distance between signals, most reference signals that are far away from the observation can safely be ignored. Thus, instead of computing the distance to all reference signals, which could become costly, we can operate on only a very samll fraction of them without significantly affecting the outcome.

6 Project Schedule and Milestones

Oct 1,2013 - Oct 31,2013
Learn Programming language: Python and C
Accomplished

Nov 1,2013 - Nov 30,2013
Write codes to classify data as different topics
Accomplished

Dec 1,2013 - Dec 30,2013
Write normalization codes
Accomplished

Jan 1,2014 - Jan 31,2014
Write parameter optimization codes
Accomplished

Feb 1,2014 - Feb 28,2014
Write streaming predictor codes
Accomplished

Mar 1,2014 - Mar 31,2014
Write Validation Codes
Accomplished

April 1,2014 - April 30,2014
Testing the implementation
Accomplished

May 1,2014 - May 13,2014
Write Documentation
Accomplished

7 Deliverables

The whole software include three packages:

- Preprocessing: map.c; convert.py; reduce.c; timebin.c; queue.h; timebin.c

- Parameter Optimization: para_opt.c
- Predictor: predict.c
- Utilities: arraylist.h; norm.c; norm_dir.c; makefile; graph_multiple_dir.py; plot_rates.py; graph_signals.py; graph_signal_picking_set.py; plot_regions_detection_times.py;
- Validation: map.py; reduce.py; get_array.py; post_process.py; libzzutil1.c; libzzutil.so; test.py

References

[1] George Chen, Stanislav Nikolov, Devavrat Shah *A Latent Source Model for Online Time Series Classification*, CIPS Conference, 2013

[2] Sitaram Asur, Bernardo A. Huberman, Gabor Szabo, and Chunyan Wang *Trends in social media: Persistence and decay*, In ICWSM, 2011.

[3] Mario Cataldi, Luigi Di Caro, and Claudio Schifanella *Emerging topic detection on twitter based on temporal and social terms evaluation*, In Proceedings of the Fifth International Conference on Weblogs and Social Media, 2011

[4] Hila Becker, Mor Naaman, and Luis Gravano *Beyond trending topics: Real- world event identification on twitter*, In ICWSM, 2011.

[5] Alon Halevy, Peter Norvig, and Fernando Pereira *The unreasonable effectiveness of data*, IEEE Intelligent Systems, 2011

[6] Yen-Hsien Lee, Chih-Ping Wei, Tsang-Hsiang Cheng, and Ching-Ting Yang *Nearest neighbor based approach to time series classification*,Decision Support Systems, 2012

[7] Juan Rodriguez and Carlos Alonso *Interval and dynamic time warping based decision trees*, In proceedings of the 2004 ACM Symposium on Applied Computing

[8] Hui Ding, Goce Trajecvski *Querying and mining of time series data: experimental comparison of distance measure*, Proceedings of the VLDB Edowment, 2008

[9] Dan Feldman, Matthew Faulkner *Scalable training of mixture models via corests*, In advances in Neural Information Processing System 24, 2011

[10] Shiva Prasad, Huahua Wang *Online dictionary learning with application to novel document detection*, In Advances in Neutral Information Processing System 25, 2012

[11] Dnaiel Hsu and Sham Kakade *Learning Mixture of spherical gaussians:Moment methods and spectral decompositions*, 2013

[12] Michael Mathioudakis and Nick Koudas. *Trend Detection over the Twitter Stream*, In procedings of the 2010 ACM SIGMOD International Conference, 2010

[13] Alex Nanopoulos, Rob Alcock *Feature-based classification of time series data*, International Journal of Computer Research, 10, 2010